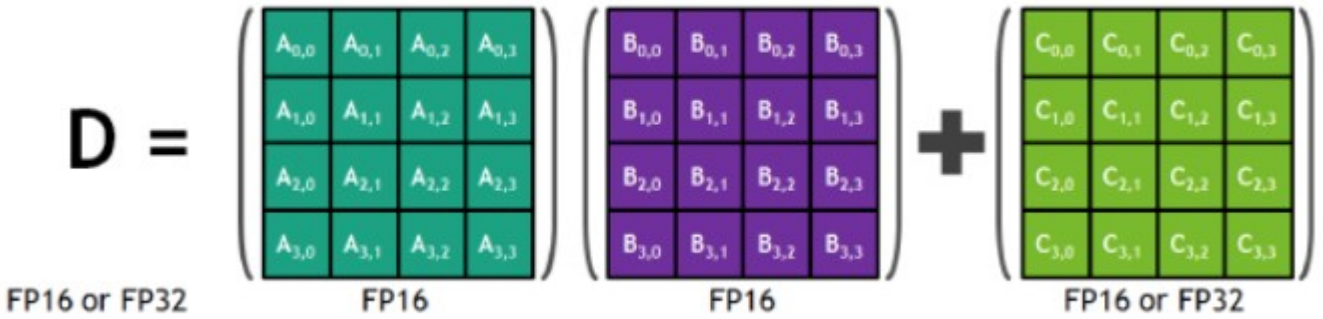
Basic Linear Algebra Operations on TensorCore GPU

Shaoshuai Zhang, Vivek Karihaloo, Panruo Wu
Department of Computer Science, University of Houston

Basic Linear Algebra Operations on TensorCore GPU

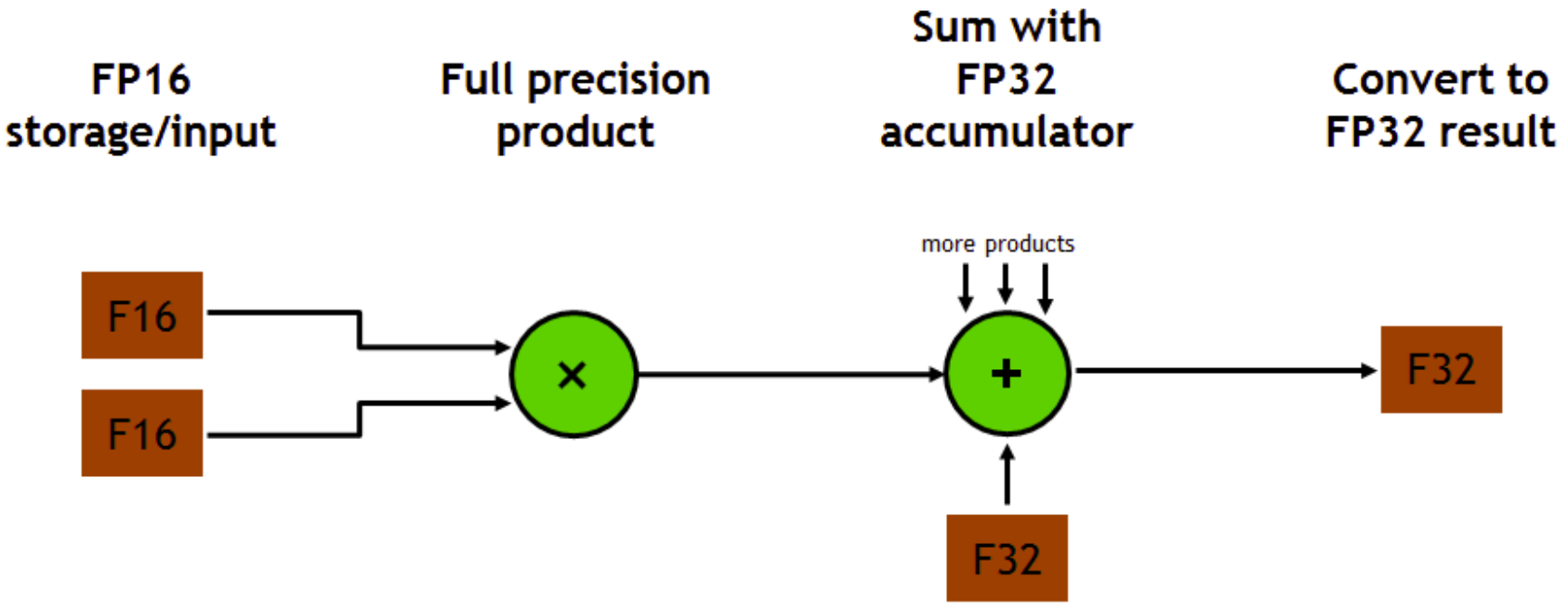
- Uses TensorCore GPU hardware
- Dense Linear Algebra operation
  - TRSM
  - SYRK
  - TRMM
- Our proposed algorithm outperforms cublas routine upto 4.7x speedup.

TensorCore GPU



- Specialized hardware to boost performance-of large dense matrix computation.
- Provides ability to perform matrix multiplication in half precision and accumulate result in full precision.
- Tensor Core performs fused multiplication addition, C = A * B + C
- Two FP16 4*4 matrices are multiplied and then the result added to 4*4 FP16 or FP32 matrix per clock cycle.

TensorCore GPU



How to program Tensor Core

- CUBLAS (distributed with CUDA)
  - Usually the fastest
  - Traditional BLAS-like API. Only supports D = A*B+C
  - Device level functions – no thread block/warp/thread level API
  - Blackbox – unknown implementation
- CUTLASS (from NVIDIA)
  - Template C++ Library
  - Almost as fast as CUBLAS
  - API support for device / thread block / warp / thread level
- WMMA C++ Intrinsic Functions
  - Not as fast as previous two on Tensor Core
  - Fairly low level, WARP level API

How is Tensor Core helpful for your application

- Design your algorithms to do matrix matrix multiplication
  - New algorithm : Blocking in dense linear algebra
  - New algorithm : Recursion usually leads to big matrix matrix multiplication
- Overkill
  - Use matrix matrix multiplication operation to perform matrix vector multiplication.
  - Or dot product
  - Most of the operation are wasted but may still be faster in the end.

BLAS operation

- BLAS 1 - vector vector computation
- BLAS 2 - matrix vector computation
- BLAS 3 – matrix matrix computation

BLAS3 routine

- TRSM - triangular solve with multiple right hand side,
  - $A = \alpha L^{-1} B$, where α represents a scalar, L represents a lower or upper triangular m × m matrix and B represents a m × n matrix.

- TRMM - triangular matrix matrix multiplication
  - triangular matrix matrix multiplication, $A = \alpha LB$, where α represents a scalar, L represents a lower or upper triangular m × m matrix and B represents a m × n matrix.

- SYRK - represents symmetric rank k update
  - $B = \alpha AA^T + \beta B$, where α and β represents a scalar, B is m×m symmetric matrix and A is a m×k matrix.

Algorithm Structure

- To take advantage of data locality –
  - Recursive vs Blocking Algorithm
  - To take advantage of TensorCore, the algorithms must be redesigned.
- Blocking algorithm
  - Uses a panel (BLAS 2 operation) and
  - Updates a trailing matrix which is a GEMM operation
- Recursive algorithm
  - It is symmetric in nature
  - Provides more and bigger GEMM’s
  - Tensorcore performs better on bigger GEMM’s
  - Hence, we use symmetric recursive algorithm for more and bigger GEMM’s.

Blocking TRSM

- Processes NB columns per iteration and then updates a trailing matrix. Performs matrix multiplication of constant size NB.

Algorithm 1 Blocking Triangular Solve with cutoff **CUTOFFSIZE**

```
1  function [X] = TRSM(A,B)
2    [m,n] = size(A);
3    nb = CUTOFFSIZE;
4    for i=1:nb:m
5       X(i:i+nb,:)=
6       PanelStrsm(A(i:i+nb,i:i+nb)
7       ,B(i:nb,:));
8       if m-i>nb
9          B(i+nb:m,:)=B(i+nb:m,:)-
10         A(i+nb:m,i:i+nb)*X(i:i+nb,:);
11      end
12   end
13 end
```

Symmetric Recursive TRSM

- Generates matrix multiplication of size NB, 2NB, …, N/2. Hence produce bigger GEMM's

Algorithm 4 Recursive Triangular Solve with cutoff **CUTOFFSIZE**

```
1  function [X] = RTRMM(A,B)
2    [m,n] = size(B);
3    if n<=CUTOFFSIZE
4      X=LeafStrsm(A(1:m, 1:m),B(1:m, :));
5      return
6    end
7    [X1] =
8    RTRSM(A(1:m/2,1:m/2),B(1:m/2, :));
9    B(m/2+1:m, :) = B(m/2+1:m, :) -
10   A(m/2+1:m,1:m/2)*X1;
11   [X2]=
12   RTRSM(A(m/2+1:m, m/2+1:m),
13   B(m/2+1:m, :));
14   X=[X1;X2];
15 end
```

MAGMA TRSM

- Non Recursive Algorithm run on GPU.
- In the leaf operation MAGMA does $A^{-1}b$ and performs GEMM operation in leaf while conventional TRSM use backward and forward substitution.
- In MAGMA STRSM routine, SGEMM are manually replaced with TC-GEMMs.

Recursive Algorithms vs. Non-Recursive Algorithms

- For recursive TRSM routine the performance is more or less the same irrespective of cut off size.
- While the performance of MAGMA TRSM and conventional blocking TRSM depends on the cut off size.

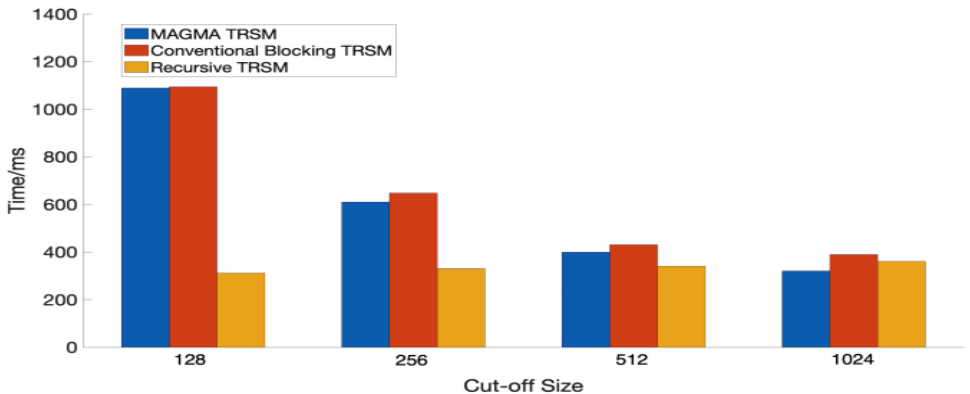


Fig. 3: Performance comparison between non-recursive TRSM and recursive TRSM. The x-axis is the cut-off size and the y-axis shows the execution time of different algorithms. 1st bar: MAGMA TRSM; 2nd bar: the conventional blocking TRSM ; 3rd bar: the proposed recursive TRSM

Recursive Algorithms vs. Non-Recursive Algorithms

- Literature suggest inv(A) * b as a linear solver is conditionally backward stable and in some cases it is not
- MAGMA TRSM
  - provides high performance but leads to unstable results.
    - In leaf implements TC-gemm
- Conventional Blocking TRSM
  - Leads to stable result but poor performance as consumes more time.
- Recursive TC TRSM
  - Provides stable results and high performance

CUT Off Size

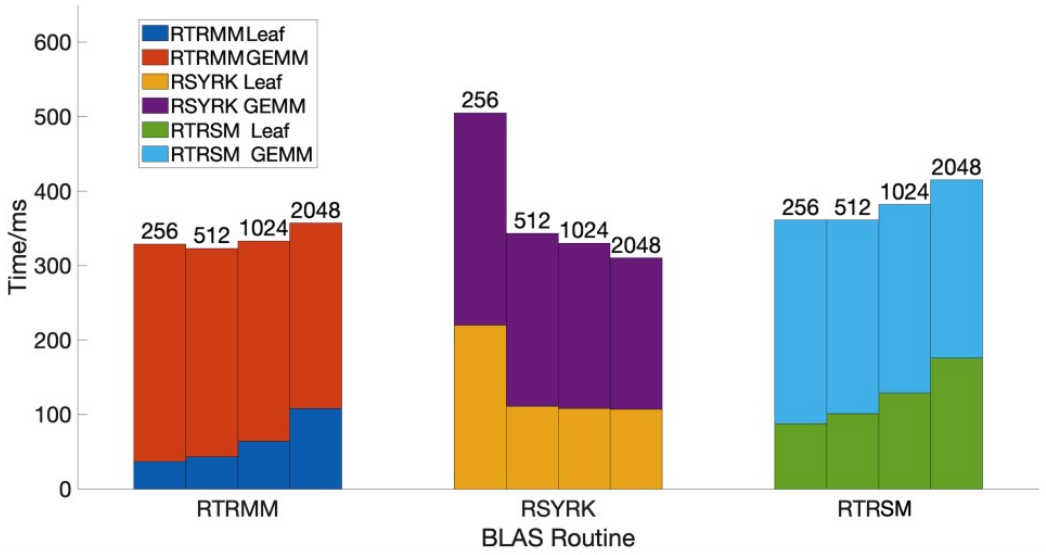


Fig. 2: The impact of different cut-off size in three recursive routines with matrix size 32768*16384. The numbers denote the cut-off size, 256-2048 from left to right.

Cut off Size

- Above graph displays performance in GEMM time plus leaf time
- Above graph for RSYRK, for block size 256 performance is poor.
  - Leaf operation uses cublas SSYRK
  - Cublas SSYRK slow when matrix is tall and skinny.
  - Can be solved by writing a kernel that optimizes it.
- Above graph shows our recursive routines are independent of cut off size
- Possible Advantage – It could be ported easily to other architectures since requires no auto tuning of parameters

Performance

- For performance analysis we have compared following algorithm
  - Enabling TensorCore in Recursive routines
  - Naive TensorCore for GEMM benchmark implementation – No recursion used
  - Disabling TensorCore in Recursive routines
  - Cublas benchmark

- For TRSM there is no comparison with Naive TC for GEMM benchmark because TRSM cannot be considered as a GEMM in context to naive TC On benchmark implementation

Performance

- Matrix size – 32768 * 16384
- . 4.7x speedup

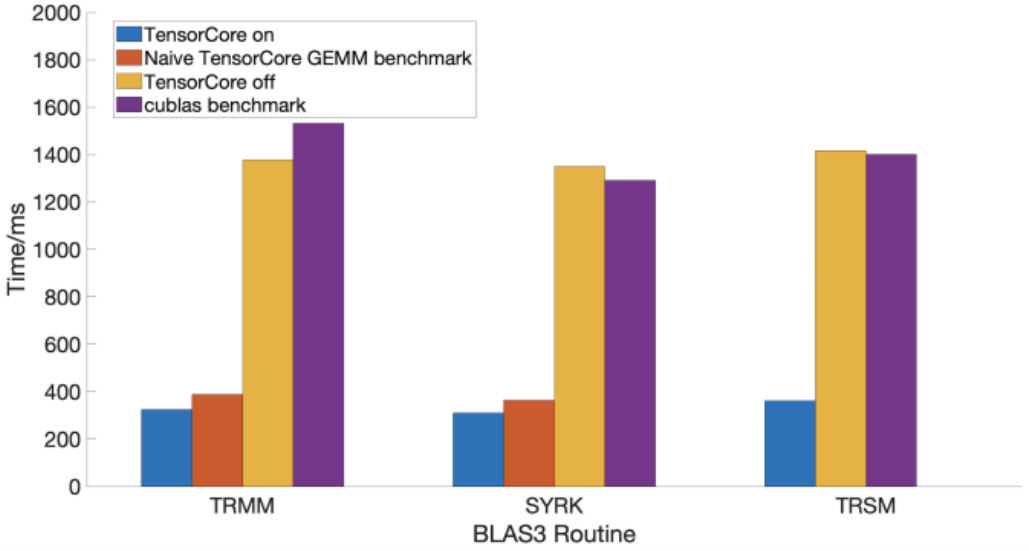


Fig. 1: Overall performance comparison between enabling TensorCore in the routines (1st bar), Naive TensorCore implementation (2nd bar), disabling TensorCore in the routines (3rd bar) and cublas corresponding benchmark (4th bar). For TRSM, the 2nd bar shows disabling TensorCore in TRSM and the 3rd bar is the cublas benchmark. The sizes of all routines are 32768*16384

Accuracy

- Accuracy will not be as accurate as cublas benchmark since they are single precision computation
- Accuracy is bounded by machine epsilon of Tensorcore
- Literature mentions TRSM is backward stable and GEMM’s are not

Routine	Forward Error	Backward Error
RTRSM		1.18E-04
RSYRK	1.19E-04	
RTRMM	2.29E-04	

Future Linear Algebra Work

- Implement LU and Cholesky algorithm with speedup on Tensorcore
- $A = LU$
  - Used to solve square system of linear equation
- $A = LL^T$ (Cholesky)
  - Hermitian positive definite matrix (eigen value positive)

QUESTIONS?